Activity Report 2017

Project-Team TEA

Time, Events and Architectures

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Embedded and Real-time Systems

Table of contents

Project-Team TEA

Creation of the Team: 2014 January 01, updated into Project-Team: 2015 January 01

Keywords:

Computer Science and Digital Science:

A1.2.7. - Cyber-physical systems
A1.5.2. - Communicating systems
A2.1.1. - Semantics of programming languages
A2.1.3. - Functional programming
A2.1.6. - Concurrent programming
A2.1.8. - Synchronous languages
A2.1.10. - Domain-specific languages
A2.2.1. - Static analysis
A2.2.4. - Parallel architectures
A2.3. - Embedded and cyber-physical systems
A2.3.1. - Embedded systems
A2.3.2. - Cyber-physical systems
A2.3.3. - Real-time systems
A2.4. - Verification, reliability, certification
A2.4.1. - Analysis
A2.4.2. - Model-checking
A2.4.3. - Proofs
A2.5. - Software engineering
A2.5.1. - Software Architecture & Design
A2.5.2. - Component-based Design
A4.4. - Security of equipment and software
A4.5. - Formal methods for security
A7.2. - Logic in Computer Science
A7.2.3. - Interactive Theorem Proving
A7.3. - Computational models
A8.1. - Discrete mathematics, combinatorics
A8.4. - Computer Algebra

Other Research Topics and Application Domains:

B5.1. - Factory of the future
B6.1.1. - Software engineering
B6.4. - Internet of things
B6.6. - Embedded systems

1. Personnel

Research Scientists

Jean-Pierre Talpin [Team leader, Inria, Senior Researcher, HDR]

Thierry Gautier [Inria, Researcher]
Rajesh Kumar Gupta [Inria, Advanced Research Position, from Aug. 2017]
Vania Joloboff [Inria, Researcher, until Sep. 2017]

PhD Students

Simon Lunel [Mitsubishi Electric]
Jean-Joseph Marty [Inria, from Oct. 2017]
Liangcong Zhang [China Scholarship Council, from Oct. 2017]

Technical staff

Loïc Besnard [CNRS SED, Senior Research Engineer, seconded at 60%]
Hai Nam Tran [Inria]
Alexandre Honorat [Inria, until Sep. 2017]

Intern

Jean-Joseph Marty [Inria, until Jun. 2017]

Administrative Assistant

Armelle Mozziconacci [CNRS]

Visiting Scientists

Alexandre Honorat [IETR, from Oct. 2017]
Deian Stefan [UC San Diego, Sep. 2016]

2. Overall Objectives

2.1. Introduction

An embedded architecture is an artifact of heterogeneous constituents and at the crossing of several design viewpoints: software, embedded in hardware, interfaced with the physical world. Time takes different forms when observed from each of these viewpoints: continuous or discrete, event-based or time-triggered. Unfortunately, modeling and programming formalisms that represent software, hardware and physics significantly alter this perception of time. Moreover, time reasoning in system design is usually isolated to a specific design problem: simulation, profiling, performance, scheduling, parallelization, simulation. The aim of project-team TEA is to define conceptually unified frameworks for reasoning on composition and integration in cyber-physical system design, and to put this reasoning to practice by revisiting analysis and synthesis issues in real-time system design with soundness and compositionality gained from formalization.

2.2. Context

In the construction of complex systems, information technology (IT) has become a central force of revolutionary changes, driven by the exponential increase of computational power. In the field of telecommunication, IT provides the necessary basis for systems of networked distributed applications. In the field of control engineering, IT provides the necessary basis for embedded control applications. The combination of telecommunication and embedded systems into networked embedded systems opens up a new range of systems, capable of providing more intelligent functionality thank to information and communication (ICT). Networked embedded systems have revolutionized several application domains: energy networks, industrial automation and transport systems.

20th-century science and technology brought us effective methods and tools for designing both computational and physical systems. But the design of cyber-physical systems (CPS) is much more than the union of those two fields. Traditionally, information scientists only have a hazy notion of requirements imposed by the physical environment of computers. Similarly, mechanical, civil, and chemical engineers view computers strictly as devices executing algorithms. To the extent we have designed CPS, we have done so in an ad hoc, on-off manner that is not repeatable. A new science of CPS design will allow us to create new machines with complex dynamics and high reliability, to apply its principles to new industries and applications in a reliable and economically efficient way. Progress requires nothing less than the construction of a new science and technology foundation for CPS that is simultaneously physical and computational.

2.3. Motivations

Beyond the buzzword, a CPS is an ubiquitous object of our everyday life. CPSs have evolved from individual independent units (e.g an ABS brake) to more and more integrated networks of units, which may be aggregated into larger components or sub-systems. For example, a transportation monitoring network aggregates monitored stations and trains through a large scale distributed system with relatively high latency. Each individual train is being controlled by a train control network, each car in the train has its own real-time bus to control embedded devices. More and more, CPSs are mixing real-time low latency technology with higher latency distributed computing technology.

In the past 15 years, CPS development has moved towards Model Driven Engineering (MDE). With MDE methodology, first all requirements are gathered together with use cases, then a model of the system is built (sometimes several models) that satisfy the requirements. There are several modeling formalisms that have appeared in the past ten years with more or less success. The most successful are the *executable* models [1] [2] [3], i.e., models that can be simulated, exercised, tested and validated. This approach can be used for both software and hardware.

A common feature found in CPSs is the ever presence of concurrency and parallelism in models. Large systems are increasingly mixing both types of concurrency. They are structured hierarchically and comprise multiple synchronous devices connected by buses or networks that communicate asynchronously. This led to the advent of so-called GALS (Globally Asynchronous, Locally Synchronous) models, or PALS (Physically Asynchronous, Logically Synchronous) systems, where reactive synchronous objects are communicating asynchronously. Still, these infrastructures, together with their programming models, share some fundamental concerns: parallelism and concurrency synchronization, determinism and functional correctness, scheduling optimality and calculation time predictability.

Additionally, CPSs monitor and control real-world processes, the dynamics of which are usually governed by physical laws. These laws are expressed by physicists as mathematical equations and formulas. Discrete CPS models cannot ignore these dynamics, but whereas the equations express the continuous behavior usually using real numbers (irrational) variables, the models usually have to work with discrete time and approximate floating point variables.

2.4. Challenges

A cyber-physical, or reactive, or embedded system is the integration of heterogeneous components originating from several design viewpoints: reactive software, some of which is embedded in hardware, interfaced with the physical environment through mechanical parts. Time takes different forms when observed from each of these viewpoints: it is discrete and event-based in software, discrete and time-triggered in hardware, continuous in mechanics or physics. Design of CPS often benefits from concepts of multiform and logical time(s) for their natural description. High-level formalisms used to model software, hardware and physics additionally alter this perception of time quite significantly.

[1] *Matlab/Simulink*, https://fr.mathworks.com/products/simulink.html
[2] *Ptolemy*, http://ptolemy.eecs.berkeley.edu
[3] *SysML*, http://www.uml-sysml.org

In model-based system design, time is usually abstracted to serve the purpose of one of many design tasks: verification, simulation, profiling, performance analysis, scheduling analysis, parallelization, distribution, or virtual prototyping. For example in non-real-time commodity software, timing abstraction such as number of instructions and algorithmic complexity is sufficient: software will run the same on different machines, except slower or faster. Alternatively, in cyber-physical systems, multiple recurring instances of meaningful events may create as many dedicated logical clocks, on which to ground modeling and design practices.

Time abstraction increases efficiency in event-driven simulation or execution (i.e SystemC simulation models try to abstract time, from cycle-accurate to approximate-time, and to loosely-time), while attempting to retain functionality, but without any actual guarantee of valid accuracy (responsibility is left to the model designer). Functional determinism (a.k.a. conflict-freeness in Petri Nets, monotonicity in Kahn PNs, confluence in Milner's CCS, latency-insensitivity and elasticity in circuit design) allows for reducing to some amount the problem to that of many schedules of a single self-timed behavior, and time in many systems studies is partitioned into models of computation and communication (MoCCs). Multiple, multiform time(s) raises the question of combination, abstraction or refinement between distinct time bases. The question of combining continuous time with discrete logical time calls for proper discretization in simulation and implementation. While timed reasoning takes multiple forms, there is no unified foundation to reasoning about multi-form time in system design.

The objective of project-team TEA is henceforth to define formal models for timed quantitative reasoning, composition, and integration in embedded system design. Formal time models and calculi should allow us to revisit common domain problems in real-time system design, such as time predictability and determinism, memory resources predictability, real-time scheduling, mixed-criticality and power management; yet from the perspective gained from inter-domain timed and quantitative abstraction or refinement relations. A regained focus on fundamentals will allow to deliver better tooled methodologies for virtual prototyping and integration of embedded architectures.

3. Research Program

3.1. Previous Works

The challenges of team TEA support the claim that sound Cyber-Physical System design (including embedded, reactive, and concurrent systems altogether) should consider multi-form time models as a central aspect. In this aim, architectural specifications found in software engineering are a natural focal point to start from. Architecture descriptions organize a system model into manageable components, establish clear interfaces between them, collect domain-specific constraints and properties to help correct integration of components during system design. The definition of a formal design methodology to support heterogeneous or multi-form models of time in architecture descriptions demands the elaboration of sound mathematical foundations and the development of formal calculi and methods to instrument them. This constitutes the research program of team TEA.

System design based on the "synchronous paradigm" has focused the attention of many academic and industrial actors on abstracting non-functional implementation details from system design. This elegant design abstraction focuses on the logic of interaction in reactive programs rather than their timed behavior, allowing to secure functional correctness while remaining an intuitive programming model for embedded systems. Yet, it corresponds to embedded technologies of single cores and synchronous buses from the 90s, and may hardly cover the semantic diversity of distribution, parallelism, heterogeneity, of cyber-physical systems found in 21st century Internet-connected, true-time^{TM}-synchronized clouds, of tomorrow's grids.

By contrast with a synchronous hypothesis yet from the same era, the polychronous MoCC implemented in the data-flow specification language Signal, available in the Eclipse project POP [4] and in the CCSL

[4] *Polychrony on Polarsys*, https://www.polarsys.org/projects/polarsys.pop

standard. [5], are inherently capable of describing multi-clock abstractions of GALS systems. The POP and TimeSquare projects provide tooled infrastructures to refine high-level specifications into real-time streaming applications or locally synchronous and globally asynchronous systems, through a series of model analysis, verification, and synthesis services. These tool-supported refinement and transformation techniques can assist the system engineer from the earliest design stages of requirement specification to the latest stages of synthesis, scheduling and deployment. These characteristics make polychrony much closer to the required semantic for compositional, refinement-based, architecture-driven, system design.

While polychrony was a step ahead of the traditional synchronous hypothesis, CCSL is a leap forward from synchrony and polychrony. The essence of CCSL is “multi-form time” toward addressing all of the domain-specific physical, electronic and logical aspects of cyber-physical system design.

3.2. Modeling Times

To make a sense and eventually formalize the semantics of time in system design, we should most certainly rely on algebraic representations of time found in previous works and introduce the paradigm of "time systems" (type systems to represent time) in a way reminiscent to CCSL. Just as a type system abstracts data carried along operations in a program, a time system abstracts the causal interaction of that program module or hardware element with its environment, its pre and post conditions, its assumptions and guarantees, either logical or numerical, discrete or continuous. Some fundamental concepts of the time systems we envision are present in the clock calculi found in data-flow synchronous languages like Signal or Lustre, yet bound to a particular model of concurrency, hence time.

In particular, the principle of refinement type systems [6], is to associate information (data-types) inferred from programs and models with properties pertaining, for instance, to the algebraic domain on their value, or any algebraic property related to its computation: effect, memory usage, pre-post condition, value-range, cost, speed, time, temporal logic [7]. Being grounded on type and domain theories, a time system should naturally be equipped with program analysis techniques based on type inference (for data-type inference) or abstract interpretation (for program properties inference) to help establish formal relations between heterogeneous component “types”. Just as a time calculus may formally abstract timed concurrent behaviors of system components, timed relations (abstraction and refinement) represent interaction among components.

Scalability and compositionality requires the use of assume-guarantee reasoning to represent them, and to facilitate composition by behavioral sub-typing, in the spirit of the (static) contract-based formalism proposed by Passerone et al. [8]. Verification problems encompassing heterogeneously timed specifications are common and of great variety: checking correctness between abstract and concrete time models relates to desynchronisation (from synchrony to asynchrony) and scheduling analysis (from synchrony to hardware). More generally, they can be perceived from heterogeneous timing viewpoints (e.g. mapping a synchronous-time software on a real-time middle-ware or hardware).

This perspective demands capabilities not only to inject time models one into the other (by abstract interpretation, using refinement calculi), to compare time abstractions one another (using simulation, refinement, bi-simulation, equivalence relations) but also to prove more specific properties (synchronization, determinism, endochrony). All this formalization effort will allow to effectively perform the tooled validation of common cross-domain properties (e.g. cost v.s. power v.s. performance v.s. software mapping) and tackle equally common yet though case studies such as these linking battery capacity, to on-board CPU performance, to static software schedulability, to logical software correctness and plant controllability: the choice of the right sampling period across the system components.

[5] *Clock Constraints in UML/MARTE CCSL*. C. André, F. Mallet. RR-6540. Inria, 2008. http://hal.inria.fr/inria-00280941
[6] *Abstract Refinement Types*. N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.
[7] *LTL types FRP*. A. Jeffrey. Programming Languages meets Program Verification.
[8] *A contract-based formalism for the specification of heterogeneous systems*. L. Benvenistu, et al. FDL, 2008

3.3. Modeling Architectures

To address the formalization of such cross-domain case studies, modeling the architecture formally plays an essential role. An architectural model represents components in a distributed system as boxes with well-defined interfaces, connections between ports on component interfaces, and specifies component properties that can be used in analytical reasoning about the model. Several architectural modeling languages for embedded systems have emerged in recent years, including the SAE AADL [9], SysML [10], UML MARTE [11].

In system design, an architectural specification serves several important purposes. First, it breaks down a system model into manageable components to establish clear interfaces between components. In this way, complexity becomes manageable by hiding details that are not relevant at a given level of abstraction. Clear, formally defined, component interfaces allow us to avoid integration problems at the implementation phase. Connections between components, which specify how components affect each other, help propagate the effects of a change in one component to the linked components.

Most importantly, an architectural model is a repository to share knowledge about the system being designed. This knowledge can be represented as requirements, design artifacts, component implementations, held together by a structural backbone. Such a repository enables automatic generation of analytical models for different aspects of the system, such as timing, reliability, security, performance, energy, etc. Since all the models are generated from the same source, the consistency of assumptions w.r.t. guarantees, of abstractions w.r.t. refinements, used for different analyses becomes easier, and can be properly ensured in a design methodology based on formal verification and synthesis methods.

Related works in this aim, and closer in spirit to our approach (to focus on modeling time) are domain-specific languages such as Prelude [12] to model the real-time characteristics of embedded software architectures. Conversely, standard architecture description languages could be based on algebraic modeling tools, such as interface theories with the ECDAR tool [13].

In project TEA, it takes form by the normalization of the AADL standard's formal semantics and the proposal of a time specification annex in the form of related standards, such as CCSL, to model concurrency time and physical properties, and PSL, to model timed traces.

3.4. Scheduling Theory

Based on sound formalization of time and CPS architectures, real-time scheduling theory provides tools for predicting the timing behavior of a CPS which consists of many interacting software and hardware components. Expressing parallelism among software components is a crucial aspect of the design process of a CPS. It allows for efficient partition and exploitation of available resources.

The literature about real-time scheduling [14] provides very mature schedulability tests regarding many scheduling strategies, preemptive or non-preemptive scheduling, uniprocessor or multiprocessor scheduling, etc. Scheduling of data-flow graphs has also been extensively studied in the past decades.

A milestone in this prospect is the development of abstract affine scheduling techniques [15]. It consists, first, of approximating task communication patterns (e.g. between Safety-Critical Java threads) using cyclo-static data-flow graphs and affine functions. Then, it uses state of the art ILP techniques to find optimal schedules and to concretize them as real-time schedules in the program implementations [16] [17].

Abstract scheduling, or the use of abstraction and refinement techniques in scheduling borrowed to the theory of abstract interpretation [18] is a promising development toward tooled methodologies to orchestrate

[9] *Architecture Analysis and Design Language*, AS-5506. SAE, 2004. http://standards.sae.org/as5506b
[10] *System modeling Language*. OMG, 2007. http://www.omg.org/spec/SysML
[11] *UML Profile for MARTE*. OMG, 2009. http://www.omg.org/spec/MARTE
[12] *The Prelude language*. LIFL and ONERA, 2012. http://www.lifl.fr/~forget/prelude.html
[13] *PyECDAR, timed games for timed specifications*. Inria, 2013. https://project.inria.fr/pyecdar
[14] *A survey of hard real-time scheduling for multiprocessor systems*. R. I. Davis and A. Burns. *ACM Computing Survey* 43(4), 2011.
[15] *Buffer minimization in EDF scheduling of data-flow graphs*. A. Bouakaz and J.-P. Talpin. LCTES, ACM, 2013.
[16] *ADFG for the synthesis of hard real-time applications*. A. Bouakaz, J.-P. Talpin, J. Vitek. ACSD, IEEE, June 2012.
[17] *Design of SCJ Level 1 Applications Using Affine Abstract Clocks*. A. Bouakaz and J.-P. Talpin. SCOPES, ACM, 2013.

thousands of heterogeneous hardware/software blocks on modern CPS architectures (just consider modern cars or aircrafts). It is an issue that simply defies the state of the art and known bounds of complexity theory in the field, and consequently requires a particular address.

To develop the underlying theory of this promising research topic, we first need to deepen the theoretical foundation to establish links between scheduling analysis and abstract interpretation. A theory of time systems would offer the ideal framework to pursue this development. It amounts to representing scheduling constraints, inferred from programs, as types or contract properties. It allows to formalize the target time model of the scheduler (the architecture, its middle-ware, its real-time system) and defines the basic concepts to verify assumptions made in one with promises offered by the other: contract verification or, in this case, synthesis.

3.5. Virtual Prototyping

Virtual Prototyping is the technology of developing realistic simulators from models of a system under design; that is, an emulated device that captures most, if not all, of the required properties of the real system, based on its specifications. A virtual prototype should be run and tested like the real device. Ideally, the real application software would be run on the virtual prototyping platform and produce the same results as the real device with the same sequence of outputs and reported performance measurements. This may be true to some extent only. Some trade-offs have often to be made between the accuracy of the virtual prototype, and time to develop accurate models.

In order to speed-up simulation time, the virtual prototype must trade-off with something. Depending upon the application designer's goals, one may be interested in trading some loss of accuracy in exchange for simulation speed, which leads to constructing simulation models that focus on some design aspects and provide abstraction of others. A simulation model can provide an abstraction of the simulated hardware in three directions:

- *Computation abstraction*. A hardware component computes a high level function by carrying out a series of small steps executed by composing logical gates. In a virtual prototyping environment, it is often possible to compute the high level function directly by using the available computing resources on the simulation host machine, thus abstracting the hardware function.
- *Communication abstraction*. Hardware components communicate together using some wiring, and some protocol to transmit the data. Simulation of the communication and the particular protocol may be irrelevant for the purpose of virtual prototyping: communication can be abstracted into higher level data transmission functions.
- *Timing Abstraction*. In a cycle accurate simulator, there are multiple simulation tasks, and each task makes some progress on each clock cycle, but this slows down the simulation. In a virtual prototyping experiment, one may not need such precise timing information: coarser time abstractions can be defined allowing for faster simulation.

The cornerstone of a virtual prototyping platform is the component that simulates the processor(s) of the platform, and its associated peripherals. Such simulation can be *static* or *dynamic*.

A solution usually adopted to handle time in virtual prototyping is to manage hierarchical time scales, use component abstractions where possible to gain performance, use refinement to gain accuracy where needed. Localized time abstraction may not only yield faster simulation, but facilitate also verification and synthesis (e.g. synchronous abstractions of physically distributed systems). Such an approach requires computations and communications to be harmoniously discretized and abstracted from originally heterogeneous viewpoints onto a structuring, articulating, pivot model, for concerted reasoning about time and scheduling of events in a way that ensures global system specification correctness.

In the short term these component models could be based on libraries of predefined models of different levels of abstractions. Such abstractions are common in large programming workbench for hardware modeling, such as SystemC, but less so, because of the engineering required, for virtual prototyping platforms.

[18] *La vérification de programmes par interprétation abstraite*. P. Cousot. Séminaire au Collège de France, 2008.

The approach of team TEA provides an additional ingredient in the form of rich component interfaces. It therefore dictates to further investigate the combined use of conventional virtual prototyping libraries, defined as executable abstractions of real hardware, with executable component simulators synthesised from rich interface specifications (using, e.g., conventional compiling techniques used for synchronous programs).

4. Application Domains

4.1. Automotive and Avionics

From our continuous collaboration with major academic and industrial partners through projects TOPCASED, OPENEMBEDD, SPACIFY, CESAR, OPEES, P and CORAIL, our experience has primarily focused on the aerospace domain. The topics of time and architecture of team TEA extend to both avionics and automotive. Yet, the research focus on time in team TEA is central in any aspect of, cyber-physical, embedded system design in factory automation, automotive, music synthesis, signal processing, software radio, circuit and system on a chip design; many application domains which, should more collaborators join the team, would definitely be worth investigating.

Multi-scale, multi-aspect time modeling, analysis and software synthesis will greatly contribute to architecture modeling in these domains, with applications to optimized (distributed, parallel, multi-core) code generation for avionics (project Corail with Thales avionics, section 8) as well as modeling standards, real-time simulation and virtual integration in automotive (project with Toyota ITC, section 8).

Together with the importance of open-source software, one of these projects, the FUI Project P (section 8), demonstrated that a centralized model for system design could not just be a domain-specific programming language, such as discrete Simulink data-flows or a synchronous language. Synchronous languages implement a fixed model of time using logical clocks that are abstraction of time as sensed by software. They correspond to a fixed viewpoint in system design, and in a fixed hardware location in the system, which is not adequate to our purpose and must be extended.

In project P, we first tried to define a centralized model for importing discrete-continuous models onto a simplified implementation of SIMULINK: P models. Certified code generators would then be developed from that format. Because this does not encompass all aspects being translated to P, the P meta-model is now being extended to architecture description concepts (of the AADL) in order to become better suited for the purpose of system design. Another example is the development of System modeler on top of SCADE, which uses the more model-engineering flavored formalism SysML to try to unambiguously represent architectures around SCADE modules.

An abstract specification formalism, capable of representing time, timing relations, with which heterogeneous models can be abstracted, from which programs can be synthesized, naturally appears better suited for the purpose of virtual prototyping. RT-Builder, based on Signal like Polychrony and developed by TNI, was industrially proven and deployed for that purpose at Peugeot. It served to develop the virtual platform simulating all on-board electronics of PSA cars. This ‘hardware in the loop” simulator was used to test equipments supplied by other manufacturers with respect to virtual cars. In the advent of the related automotive standard, RT-Builder then became AUTOSAR-Builder.

4.2. Factory Automation

In collaboration with Mitsubishi R&D, we explore another application domain where time and domain heterogeneity are prime concerns: factory automation. In factory automation alone, a system is conventionally built from generic computing modules: PLCs (Programmable Logic Controllers), connected to the environment with actuators and detectors, and linked to a distributed network. Each individual, physically distributed, PLC module must be timely programmed to perform individually coherent actions and fulfill the global physical, chemical, safety, power efficiency, performance and latency requirements of the whole production chain. Factory chains are subject to global and heterogeneous (physical, electronic, functional) requirements whose enforcement must be orchestrated for all individual components.

Model-based analysis in factory automation emerges from different scientific domains and focus on different CPS abstractions that interact in subtle ways: logic of PLC programs, real-time electromechanical processing, physical and chemical environments. This yields domain communication problems that render individual domain analysis useless. For instance, if one domain analysis (e.g. software) modifies a system model in a way that violates assumptions made by another domain (e.g. chemistry) then the detection of its violation may well be impossible to explain to either of the software and chemistry experts. As a consequence, cross-domain analysis issues are discovered very late during system integration and lead to costly fixes. This is particularly prevalent in multi-tier industries, such as avionic, automotive, factories, where systems are prominently integrated from independently-developed parts.

5. Highlights of the Year

5.1. Highlights of the Year

Inria created a new International Chair and appointed American computer engineer Rajesh Gupta to the part-time position. Gupta is a professor and former chair of the Computer Science and Engineering (CSE) department in the Jacobs School of Engineering at the University of California San Diego. Rajesh Gupta will hold the International Chair for a period of five years. Starting this summer, he will engage with researchers in Inria's research center in Rennes. The position enables him to spend as much as a year spread out over the five years of his appointment.

6. New Software and Platforms

6.1. ADFG

Affine data-flow graphs schedule synthesizer
KEYWORDS: Code generation - Scheduling - Static program analysis
FUNCTIONAL DESCRIPTION: ADFG is a synthesis tool of real-time system scheduling parameters: ADFG computes task periods and buffer sizes of systems resulting in a trade-off between throughput maximization and buffer size minimization. ADFG synthesizes systems modeled by ultimately cyclo-static dataflow (UCSDF) graphs, an extension of the standard CSDF model.

Knowing the WCET (Worst Case Execute Time) of the actors and their exchanges on the channels, ADFG tries to synthezise the scheduler of the application. ADFG offers several scheduling policies and can detect unschedulable systems. It ensures that the real scheduling does not cause overflows or underflows and tries to maximize the throughput (the processors utilization) while minimizing the storage space needed between the actors (i.e. the buffer sizes).

Abstract affine scheduling is first applied on the dataflow graph, that consists only of periodic actors, to compute timeless scheduling constraints (e.g. relation between the speeds of two actors) and buffering parameters. Then, symbolic schedulability policies analysis (i.e., synthesis of timing and scheduling parameters of actors) is applied to produce the scheduller for the actors.

ADFG, initially defined to synthesize real-time schedulers for SCJ/L1 applications, may be used for scheduling analysis of AADL programs.

- Authors: Thierry Gautier, Jean-Pierre Talpin, Adnan Bouakaz, Alexandre Honorat and Loïc Besnard
- Contact: Loïc Besnard

6.2. POLYCHRONY

KEYWORDS: Code generation - AADL - Proof - Optimization - Multi-clock - GALS - Architecture - Cosimulation - Real time - Synchronous Language

FUNCTIONAL DESCRIPTION: Polychrony is an Open Source development environment for critical/embedded systems. It is based on Signal, a real-time polychronous data-flow language. It provides a unified model-driven environment to perform design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony. It can be included in heterogeneous design systems with various input formalisms and output languages. The Polychrony tool-set provides a formal framework to: validate a design at different levels, by the way of formal verification and/or simulation, refine descriptions in a top-down approach, abstract properties needed for black-box composition, compose heterogeneous components (bottom-up with COTS), generate executable code for various architectures. The Polychrony tool-set contains three main components and an experimental interface to GNU Compiler Collection (GCC):

* The Signal toolbox, a batch compiler for the Signal language, and a structured API that provides a set of program transformations. Itcan be installed without other components and is distributed under GPL V2 license.

* The Signal GUI, a Graphical User Interface to the Signal toolbox (editor + interactive access to compiling functionalities). It can be used either as a specific tool or as a graphical view under Eclipse. It has been transformed and restructured, in order to get a more up-to-date interface allowing multi-window manipulation of programs. It is distributed under GPL V2 license.

* The POP Eclipse platform, a front-end to the Signal toolbox in the Eclipse environment. It is distributed under EPL license.

- Participants: Loïc Besnard, Paul Le Guernic and Thierry Gautier
- Partners: CNRS - Inria
- Contact: Loïc Besnard
- URL: https://www.polarsys.org/projects/polarsys.pop

6.3. Polychrony AADL2SIGNAL

KEYWORDS: Real-time application - Polychrone - Synchronous model - Polarsys - Polychrony - Signal - AADL - Eclipse - Meta model
FUNCTIONAL DESCRIPTION: This polychronous MoC has been used previously as semantic model for systems described in the core AADL standard. The core AADL is extended with annexes, such as the Behavior Annex, which allows to specify more precisely architectural behaviors. The translation from AADL specifications into the polychronous model should take into account these behavior specifications, which are based on description of automata.

For that purpose, the AADL state transition systems are translated as Signal automata (a slight extension of the Signal language has been defined to support the model of polychronous automata).

Once the AADL model of a system transformed into a Signal program, one can analyze the program using the Polychrony framework in order to check if timing, scheduling and logical requirements over the whole system are met.

We have implemented the translation and experimented it using a concrete case study, which is the AADL modeling of an Adaptive Cruise Control (ACC) system, a highly safety-critical system embedded in recent cars.

- Participants: Huafeng Yu, Loïc Besnard, Paul Le Guernic, Thierry Gautier and Yue Ma
- Partner: CNRS
- Contact: Loïc Besnard
- URL: http://www.inria.fr/equipes/tea

6.4. POP

Polychrony on Polarsys

KEYWORDS: Synchronous model - Model-driven engineering
FUNCTIONAL DESCRIPTION: The Eclipse project POP is a model-driven engineering front-end to our open-source toolset Polychrony. a major achievement of the ESPRESSO (and now TEA) project-team. The Eclipse project POP is a model-driven engineering front-end to our open-source toolset Polychrony. It was finalised in the frame of project OPEES, as a case study: by passing the POLARSYS qualification kit as a computer aided simulation and verification tool. This qualification was implemented by CS Toulouse in conformance with relevant generic (platform independent) qualification documents. Polychrony is now distributed by the Eclipse project POP on the platform of the POLARSYS industrial working group. Team TEA aims at continuing its dissemination to academic partners, as to its principles and features, and industrial partners, as to the services it can offer.

Project POP is composed of the Polychrony tool set, under GPL license, and its Eclipse framework, under EPL license. SSME (Syntactic Signal-Meta under Eclipse), is the meta-model of the Signal language implemented with Eclipse/Ecore. It describes all syntactic elements specified in Signal Reference Manual [19]: all Signal operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration). The meta-model primarily aims at making the language and services of the Polychrony environment available to inter-operation and composition with other components (e.g. AADL, Simulink, GeneAuto, P) within an Eclipse-based development tool-chain. Polychrony now comprises the capability to directly import and export Ecore models instead of textual Signal programs, in order to facilitate interaction between components within such a tool-chain. The download site for project POP has opened in 2015 at https://www.polarsys.org/projects/polarsys.pop. It should be noted that the Eclipse Foundation does not host code under GPL license. So, the Signal toolbox useful to compile Signal code from Eclipse is hosted on our web server.

- Participants: Jean-Pierre Talpin, Loïc Besnard, Paul Le Guernic and Thierry Gautier
- Contact: Loïc Besnard
- URL: https://www.polarsys.org/projects/polarsys.pop

6.5. Sigali

FUNCTIONAL DESCRIPTION: Sigali is a model-checking tool that operates on ILTS (Implicit Labeled Transition Systems, an equational representation of an automaton), an intermediate model for discrete event systems. It offers functionalities for verification of reactive systems and discrete controller synthesis. The techniques used consist in manipulating the system of equations instead of the set of solutions, which avoids the enumeration of the state space. Each set of states is uniquely characterized by a predicate and the operations on sets can be equivalently performed on the associated predicates. Therefore, a wide spectrum of properties, such as liveness, invariance, reachability and attractivity, can be checked. Algorithms for the computation of predicates on states are also available. Sigali is connected with the Polychrony environment (Tea project-team) as well as the Matou environment (VERIMAG), thus allowing the modeling of reactive systems by means of Signal Specification or Mode Automata and the visualization of the synthesized controller by an interactive simulation of the controlled system.

- Contact: Hervé Marchand

7. New Results

7.1. ADFG: Affine data-flow graphs scheduler synthesis

Participants: Loïc Besnard, Thierry Gautier, Alexandre Honorat, Jean-Pierre Talpin, Hai Nam Tran.

19

SIGNAL V4-Inria version: Reference Manual. Besnard, L., Gautier, T. and Le Guernic, P. http://www.irisa.fr/espresso/Polychrony, 2010

We consider with ADFG (Affine DataFlow Graph) the synthesis of periodic scheduling parameters for real-time systems modeled as ultimately cyclo-static dataflow (UCSDF) graphs [14]. This synthesis aims for a trade-off between throughput maximization and total buffer size minimization. The synthesizer inputs are: a UCSDF graph which describes tasks by their Worst Case Execution Time (WCET), and directed buffers connecting tasks by their data production and consumption rates; the number of processors in the target system and the real-time scheduling synthesis algorithm to be used. The outputs are the synthesized scheduling parameters: the tasks periods, offsets, processor bindings and priorities, and the buffers initial marking and maximum sizes.

ADFG was originally the implementation of Adnan Bouakaz's work [20]. However the tool had not been packaged yet to be easily installed and used. Moreover, code refactoring led to improve the theory, and to add new features. Firstly, more accurate bounds and Integer Linear Programming (ILP) formulations have been used. Besides, dataflow graphs do not need to be weakly connected for EDF policy on multiprocessor systems. The new implementation also avoids to use a fixed parameter for some multiprocessor partitioning algorithms, now an optional strategy enables to compute it. Finally implementation has been adapted to standard technologies to be more easily installed and used. As the synthesizer evolved a lot, new evaluations have been made. Moreover, many scheduled examples have been simulated with Cheddar [21], which provides pertinent metrics to analyze the scheduling efficiency.

ADFG is being extended to investigate and solve the scheduling problem of dataflow programs on many-core architectures. These architectures have distinctive traits requiring significant changes to classical multiprocessor scheduling theory. There is a high number of contention points introduced by novel memory architectures and new interconnect types such as Network-on-Chip. Two solutions are proposed and implemented in ADFG: contention-aware and contention-free scheduling synthesis. We either take into account the contention and synthesize a contention-aware schedule or find a schedule that results in no contention.

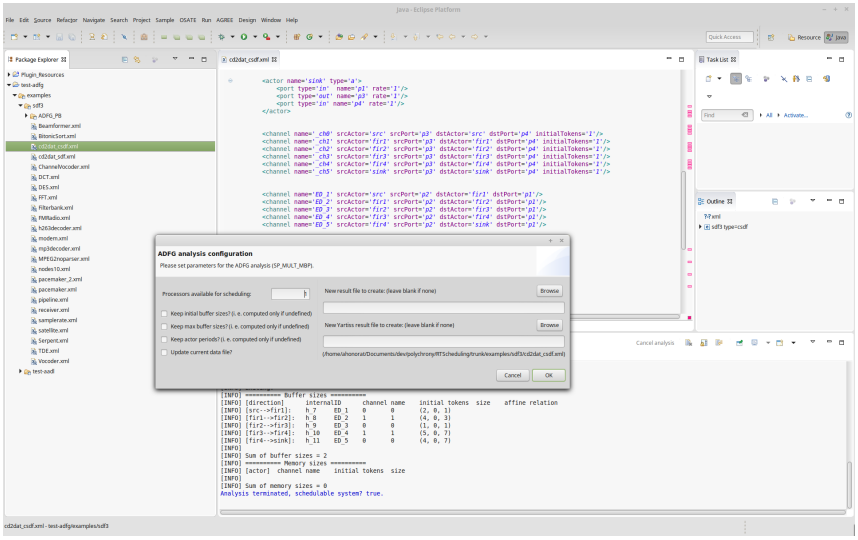

Figure 1. ADFG under Eclipse

[20] *Real-Time Scheduling of Dataflow Graphs.* A. Bouakaz. PhD Thesis, University of Rennes 1, 2013.
[21] *The Cheddar project: a GPL real-time scheduling analyzer.* http://beru.univ-brest.fr/~singhoff/cheddar/

7.2. Formal Semantics of Behavior Specifications in the Architecture Analysis and Design Language Standard

Participants: Loïc Besnard, Thierry Gautier, Jean-Pierre Talpin.

The Architecture Analysis and Design Language (AADL) is a standard proposed by SAE to express architecture specifications and share knowledge between the different stakeholders about the system being designed. To support unambiguous reasoning, formal verification, high-fidelity simulation of architecture specifications in a model-based AADL design workflow, we have defined formal semantics for the behavior specification of the AADL. These semantics rely on the structure of automata present in the standard already, yet provide tagged, trace semantics framework to establish formal relations between (synchronous, timed, asynchronous) usages or interpretations of behavior [17]. We define the model of computation and communication of a behavior specification by the synchronous, timed or asynchronous traces of automata with variables. These constrained automata are derived from *polychronous automata* defined within the polychronous model of computation and communication [11].

States of a behavior annex transition system can be either observable from the outside (*initial*, *final* or *complete* states), that is states in which the execution of the component is paused or stopped and its outputs are available; or non observable execution states, that is internal states. We thus define two kinds of steps in the transition system: *small steps*, that is non-observable steps from or to an internal state; and *big steps*, that is observable steps from a *complete* state to another, through a number of small steps). The semantics of the AADL considers the observable states of the automaton. The set of states S_A of automaton A (used to interpret the behavior annex) thus only contains states corresponding to these observable states and the set of transitions T_A big-step transitions from an observable state to another (by opposition with small-step transitions from or to an execution state). The action language of the behavior annex defines actions performed during transitions. Actions associated with transitions are action blocks that are built from basic actions and a minimal set of control structures (sequences, sets, conditionals and loops). Typically, a behavior action sequence is represented by concatenating the transition systems of its elements; a behavior action set is represented by composing the transition systems of its elements.

The polychronous model of computation had been used previously as semantic model for systems described in the core AADL standard. This translation of AADL specifications into the polychronous model now takes into account the behavior specifications. The import of AADL behavior annexes (AADL-BA) to the polychronous model relies on polychronous automata and on small steps/big steps semantics. Small steps may be viewed as an implicit oversampling of the big steps. To express such implicit upsampling, a model of *Signal-thread* has been introduced in Polychrony (refer to Section "New trends and developments in Polychrony"). In that context, the translation of a behavior annex associated with an AADL thread consists mainly in the production of the corresponding Signal automaton, which is declared as a Signal-thread, and the definition of the environment required for this Signal-thread. In particular, the signal *complete-thread* is defined so that it will occur when the next state of the automaton is a *complete* state (the control will return to the scheduler): in other words, it specifies the end of a sequence of small steps.

A specific difficulty in the translation of AADL-BA is the translation of the action language, which is related to the general problem of the translation of a sequential language to a dataflow one. First, in AADL-BA actions, a given variable may be assigned several times in a sequence (for example, $x = a + b; x = x + a$). Thus an AADL-BA action has to be transformed into a SSA (static single assignment) form ($x_0 = a + b; x = x_0 + a$ in the previous example). Another possible problem is the translation of AADL-BA loop structures (for, while, do until). In our case, this is solved, again, by considering them as Signal-threads: the *dispatch-thread* event is defined by the upperbound of the clocks of the inputs of the loop and the *complete-thread* event defines the termination of the loop.

7.3. New trends and developments in Polychrony

Participants: Loïc Besnard, Thierry Gautier.

The synchronous modeling paradigm provides strong correctness guarantees for embedded system design while requiring minimal environmental assumptions. In most related frameworks, global execution correctness is achieved by ensuring the insensitivity of (logical) time in the program from (real) time in the environment. This property, called endochrony, can be statically checked, making it fast to ensure design correctness. Unfortunately, it is not preserved by composition, which makes it difficult to exploit with component-based design concepts in mind.

It has been shown that compositionality can be achieved by weakening the objective of endochrony: a weakly endochronous system is a deterministic system that can perform independent computations and communications in any order as long as this does not alter its global state. Moreover, the non-blocking composition of weakly endochronous processes is isochronous, which means that the synchronous and asynchronous compositions of weakly endochronous processes accept the same behaviors. Unfortunately, testing weak endochrony needs state-space exploration, which is very costly in the general case. Then, a particular case of weak endochrony, called polyendochrony, was defined, which allows static checking thanks to the existing clock calculus. The clock hierarchy of a polyendochronous system may have several trees, with synchronization relations between clocks placed in different trees, but the clock expressions of the clock system must be such that there is no clock expression (especially, no root clock expression) defined by symmetric difference: root clocks cannot refer to absence. In other words, the clock system must be in disjunctive form [9].

We have now implemented code generation for polyendochronous systems in Polychrony. This generation reuses techniques of distributed code generation, with rendez-vous management for synchronization constraints on clocks which are not placed in the same tree of clocks. For such a synchronization constraint $c_1 = c_2$, nodes *send* and *receive* are added in the graph, associated with clocks c_1 and c_2: for c_1, *send*(c_1) is followed by *receive*(c_2), followed itself by all the other nodes associated with clock c_1; and symmetrically for c_2. Then the subgraphs corresponding respectively to the trees where c_1 and c_2 are placed are separated, as if they were distributed on different processors. In this way, nodes *send* and *receive* become respectively outputs and inputs (both for c_1 and c_2) of the subgraphs. Finally, a communication library (MPI) is used for simulation. The following restriction is considered in the current implementation: the roots of the trees of c_1 and c_2 must be free variables.

We have also considered another extension related to clocks, again for making code generation possible for more programs than it was the case before. A characteristic of the Signal language is that it allows to specify programs which have internal accelerations with respect to their inputs and outputs. However, the constraint that implemented programs, for which code was generated, should be endochronous, restricted more or less these programs to have one single such acceleration (or clock upsampling). To abstract from this restriction, we have defined a model of so-called *Signal-thread*, that helps to confine such accelerations, and thus to generate code for programs with multiple clock upsampling. A Signal-thread is a Signal process with internal implicit upsampling; it has a *dispatch-thread* input event and a *complete-thread* output event; its outputs are delayed compared with its inputs. As the Signal-thread represents an upsampling, the *step* (see [1]) of the corresponding generated code is a loop. Such Signal-threads may be considered as a pragmatic way to implement *clock domains*.

7.4. Modular verification of cyber-physical systems using contract theory

Participants: Jean-Pierre Talpin, Benoit Boyer, David Mentre, Simon Lunel.

The primary goal of our project, in collaboration with Mitsubishi Electronics Research Centre Europe (MERCE), is to ensure correctness-by-design in realistic cyber-physical systems, i.e., systems that mix software and hardware in a physical environment, e.g., Mitsubishi factory automation lines or water-plant factory. To achieve that, we develop a verification methodology based on decomposition into components enhanced with contract reasoning.

The work of A. Platzer on Differential Dynamic Logic ($d\mathcal{L}$) holds our attention [22]. This a formalism built on the Dynamic Logic of V. Pratt augmented with the possibility of expressing Ordinary Differential Equations (ODEs), which are the usual way to model physical behaviors in physics. Combined with the ability of Dynamic Logic to specify and verify hybrid programs, $d\mathcal{L}$ is particularly fit model cyber-physical systems. The proof system associated with the logic is implemented into the theorem prover KeYmaera X. Aimed toward automatisation, it is a promising tool to spread formal methods into industry.

We have defined a syntactic parallel composition operator in $d\mathcal{L}$ which enjoys associativity and commutativity [15]. Commutativity allows to compose component in every possible order. Associativity is mandatory to modularly design a system; it allow to upgrade a system by adding new components. We have then characterized the conditions under which we can derive automatically a proof of the contract of our composition of two components, given the proof of the contract for each components. Theses theoretical results have been exemplified with an example of a cruise-controller entirely proved within the interactive theorem prover KeYmaera X.

The study of the cruise-controller example and of a water-tank system highlights some limitations of our approach. We can not handle retro-action and we have to compose in parallel components which have to be sequenced, e.g. a sensor and a computer. We have overcomed theses limitations by introducing a sequential composition operator which enjoys associativity and distributivity over the parallel composition operator. We believe it is a first step toward a composition algebra in $d\mathcal{L}$. This operator also satisfy the property that we can automatically derive a proof of the contract of our composition of two components, given the proof of the contract for each components, but under some relaxed conditions. We believe it is the first step toward a composition algebra.

Thanks to these results, a wide variety of systems are now possible to modularly design in $d\mathcal{L}$. To validate our approach, we are currently working on the implementation of our parallel composition operator as a tactic in KeYmaera X.

To challenge our ideas, we are working in the proof of a realistic cyber-physical system, a power-train system used in automotive. We plan to use it as a basis to test abstraction mechanisms to ultimately allow mix between top-down and bottom-up design.

7.5. Parametric verification of time synchronization protocols

Participants: Ocan Sankur, Jean-Pierre Talpin.

In the context of the associate-team COMPOSITE, we addressed the verification one of the apparently simplest services in any loosely-coupled distributed system : the time service. In many instances of such systems, traffic and power grids, banking and transaction networks, the accuracy and reliability of this service are critical.

In the instance of sensor networks, it is of particular interest to verify the robustness of such protocols to variations caused by the environment. Lake of power, varying temperatures, imperfect hardware, are sources of local drifts and jitters in time measurement that require self-calibration and fault-tolerance to reach distributed consensus. FTSP, the flooding time synchronization protocol, provides fault-tolerance and enables time synchronization.

In [16], we introduce an environment abstraction technique and an incremental model checking technique to prove that FTSP eventually elects a leader for any network topology and configuration (anonymized identifiers), up to a diameter $N = 7$ (with synchronous communications) and $N = 5$ (desynchronized communications), resulting in significant improvements over previous results.

7.6. Modular analysis and verification of system libraries

Participants: Jean-Joseph Marty, Jean-Pierre Talpin.

[22] *Differential Dynamic Logic for Hybrid Systems*, André Platzer, http://symbolaris.com/logic/dL.html

We are starting to develop a new perspective on the active topic of information flow control (IFC). We plan to adapt current investigations to tagged multi-core architecture, including software (virtual machines) and hardware (the Risc V processor) experiments and applications. All this work is based on the previous experience about verified Unikernel programming on low resources processors such as the Arduino (Marty's Master internship). We will define formally relations between processes and blocks of code inside a concurrent environment. This line of work will be investigated for both embedded IoT applications and cloud computing. By working with IFC at processor level and system level, we will enforce strong security foundation and focus on constraint solving analysed software.

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Grants with Industry

8.1.1. Mitsubishi Electric R&D Europe (2015-2018)

Title: Analysis and verification for correct by construction orchestration in automated factories

Inria principal investigator: Jean-Pierre Talpin, Simon Lunel

International Partner: Mitsubishi Electric R&D Europe

Duration: 2015 - 2018

Abstract: The primary goal of our project is to ensure correctness-by-design in cyber-physical systems, i.e., systems that mix software and hardware in a physical environment, e.g., Mitsubishi factory automation lines. We develop a component-based approach in Differential Dynamic Logic allowing to reason about a wide variety of heterogeneous cyber-physical systems. Our work provides tools and methodology to design and prove a system modularly.

9. Partnerships and Cooperations

9.1. National Initiatives

9.1.1. ANR

Program: ANR

Project acronym: **Feever**

Project title: Faust Environment Everyware

Duration: 2014-2016

Coordinator: Pierre Jouvelot, Mines ParisTech

Other partners: Grame, Inria Rennes, CIEREC

URL: http://www.feever.fr

Abstract: The aim of project FEEVER is to ready the Faust music synthesis language for the Web. In this context, we collaborate with Mines ParisTech to define a type system suitable to model music signals timed at multiple rates and to formally support playing music synthesized from different physical locations.

9.1.2. PAI

Program: PAI/CORAC

Project acronym: CORAIL

Project title: Composants pour l'Avionique Modulaire Étendue

Duration: July 2013 - May 2017

Coordinator: Thales Avionics

Other partners: Airbus, Dassault Aviation, Eurocopter, Sagem...

Abstract: The CORAIL project aims at defining components for Extended Modular Avionics. The contribution of project-team TEA is to define a specification method and to provide a generator of multi-task applications.

9.2. International Initiatives

9.2.1. Inria International Labs

9.2.1.1. SACCADES

Title: Saccades

International Partner:

LIAMA

East China Normal University

Inria project-teams Aoste and Tea

Duration: 2003 - now

The SACCADES project is a LIAMA project hosted by East China Normal University and jointly led by Vania Joloboff (Inria) and Min Zhang (ECNU). The SACCADES project aims at improving the development of reliable cyber physical systems and more generally of distributed systems combining asynchronous with synchronous aspects, with different but complementary angles:

- develop the theoretical support for Models of Computations and Communications (MoCCs) that are the fundamentals basis of the tools.
- develop software tools (a) to enable the development and verification of executable models of the application software, which may be local or distributed and (b) to define and optimize the mapping of software components over the available resources.
- develop virtual prototyping technology enabling the validation of the application software on the target hardware platform.

The ambition of SACCADES project is to develop

- Theoretical Support for Cyber Physical Systems
- Software Tools for design and validation of CPS
- Virtual Prototyping of CPS

9.2.2. Inria Associate Teams

9.2.2.1. Composite

Title: Compositional System Integration

International Partner (Institution - Laboratory - Researcher):

- University of California, San Diego (United States) - Microelectronic Embedded Systems Laboratory - Rajesh Gupta

Start year: 2017

See also: http://www.irisa.fr/prive/talpin/composite

Most applications that run somewhere on the internet are not optimized to do so. They execute on general purpose operating systems or on containers (virtual machines) that are built with the most conservative assumptions about their environment. While an application is specific, a large part of the system it runs on is unused, which is both a cost (to store and execute) and a security risk (many entry points).

A unikernel, on the contrary, is a system program object that only contains the necessary the operating system services it needs for execution. A unikernel is build from the composition of a program, developed using high-level programming language, with modules of a library operating system (libOS), to execute directly on an hypervisor. A unikernel can boot in milliseconds to serve a request and shut down, demanding minimal energy and resources, offering stealthiest exposure time and surface to attacks, making them the ideal platforms to deploy on sensor networks, networks of embedded devices, smart grids and clouds.

The goal of COMPOSITE is to develop the mathematical foundations for sound and efficient composition in system programming: analysis, verification and optimization technique for modular and compositional hardware-system-software integration of unikernels. We intend to further this development with the prospect of an end-to-end co-design methodology to synthesize lean and stealth networked embedded devices.

9.2.3. Inria International Partners

9.2.3.1. Convex

Title: Compositional Verification of Cyber-Physical Systems

International Partner:

- Chinese Academy of Science, Institute of Software
- Beihang University
- Nanhang University
- Nankai University

Duration: 2017 - now

Formal modeling and verification methods have successfully improved software safety and security in vast application domains in transportation, production and energy. However, formal methods are labor-intensive and require highly trained software developers. Challenges facing formal methods stem from rapid evolution of hardware platforms, the increasing amount and cost of software infrastructures, and from the interaction between software, hardware and physics in networked cyber-physical systems.

Automation and expressivity of formal verification tools must be improved not only to scale functional verification to very large software stacks, but also verify non-functional properties from models of hardware (time, energy) and physics (domain). Abstraction, compositionality and refinement are essential properties to provide the necessary scalability to tackle the complexity of system design with methods able to scale heterogeneous, concurrent, networked, timed, discrete and continuous models of cyber-physical systems.

Project Convex wants to define a CPS architecture design methodology that takes advantage of existing time and concurrency modeling standards (MARTE, AADL, Ptolemy, Matlab), yet focuses on interfacing heterogeneous and exogenous models using simple, mathematically-defined structures, to achieve the single goal of correctly integrating CPS components.

9.3. International Research Visitors

9.3.1. Visits of International Scientists

Rajesh Gupta visited project-team TEA in August and gave two 68NQTR seminars on “Building Computing Machines That Sense, Adapt and Approximate” and on “Compositional Synthesis for High-level Design of System-Chips”.

Deian Stefan visited project-team TEA in September and gave a 68NQTR seminar on “Practical multi-core information flow control”

Shuvra Bhattacharyya visited project-team TEA in August and December and gave a 68NQTR seminar on “The DSPCAD Framework for Dataflow-based Design and Implementation of Signal Processing Systems“

9.3.2. Visits to International Teams

Jean-Pierre Talpin visited UC San Diego and UC Berkeley in the context of the associate-project Composite in June.

In the context of the IIP Convex, Jean-Pierre Talpin was invited at Beihang and Nanhang Universities in April, visited Beihang and Nankai Universities in July, and Beihang, Nankai and ECNU in November, to give seminars and a introductory course on model checking.

Jean-Pierre Talpin gave an invited talk on “Parametric model-checking the FTSP protocol ” at TU Wien June 30.

Simon Lunel visited CMU and UC San Diego in December to give seminars on “compositional proofs in differential dynamic logic”.

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific events organisation

10.1.1.1. General chair, scientific chair

Jean-Pierre Talpin served as General Chair and Finance Chair of the 15th. ACM-IEEE Conference on Methods and Models for System Design in Vienna.

10.1.1.2. Member of the organizing committees

Jean-Pierre Talpin is a member of the steering committee of the ACM-IEEE Conference on Methods and Models for System Design (MEMOCODE).

10.1.2. Scientific Events Selection

10.1.2.1. Member of the Conference Program Committees

Jean-Pierre Talpin served the program committee of:

- ACSD’17, 17th. International Conference on Application of Concurrency to System Design
- LCTES’17, 20th. ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems
- SAC’18, 33rd. ACM SIGAPP Symposium on Applied Computing
- SCOPES’17, 20th. International Workshop on Software and Compilers for Embedded Systems

10.1.3. Journal

10.1.3.1. Member of the Editorial Boards

Jean-Pierre Talpin is Associate Editor with the ACM Transactions for Embedded Computing Systems (TECS).

10.1.3.2. Reviews

Thierry Gautier reviewed articles for *Information Processing Letters*.

10.1.4. Invited Talks

Jean-Pierre Talpin gave a keynote speech, entitled “Compositional methods for CPS design” at the Symposium on Dependable Software Engineering (SETTA’17) in Changsha, October 25.

10.1.5. Invited Talks

Albert Benveniste and Thierry Gautier previewed a seminar at SYNCHRON'17 to be given at the Collège de France in Gérard Berry's 2017-2018 lecture course.

10.1.6. Scientific Expertise

Jean-Pierre Talpin was nominated vice-president of ANR evaluation committee CES-25

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Jean-Pierre Talpin gave a one week class "Introduction to model-checking" at Nankai University in July and at Beihang University in November.

10.2.2. Supervision

Jean-Pierre Talpin co-supervises the PhD Thesis of Simon Lunel, Liangcong Zhang and Jean-Joseph Marty

10.2.3. Juries

Jean-Pierre Talpin served as rapporteur at the HDR Thesis defense of Jérôme Hugues, entitled "Architecture in the Service of Real-Time Middleware, Contributions to Architecture Description Languages", which took place at INP Toulouse on February 22.

Jean-Pierre Talpin served as rapporteur at the HDR Thesis defense of Maxime Pelcat, entitled "Models, methods and tools for bridging the design productivity gap of embedded signal processing systems", which took place at Institut Poincarré in Clermond-Ferrand on July 10.

11. Bibliography

Major publications by the team in recent years

[1] L. BESNARD, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Compilation of Polychronous Data Flow Equations*, in "Synthesis of Embedded Software", S. K. SHUKLA, J.-P. TALPIN (editors), Springer, 2010, pp. 1-40 [*DOI :* 10.1007/978-1-4419-6400-7_1], http://hal.inria.fr/inria-00540493

[2] A. BOUAKAZ, J.-P. TALPIN. *Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks*, in "International Workshop on Software and Compilers for Embedded Systems", St. Goar, Germany, June 2013, pp. 58-67 [*DOI :* 10.1145/2463596.2463600], https://hal.inria.fr/hal-00916487

[3] C. BRUNETTE, J.-P. TALPIN, A. GAMATIÉ, T. GAUTIER. *A metamodel for the design of polychronous systems*, in "The Journal of Logic and Algebraic Programming", 2009, vol. 78, n^{o} 4, pp. 233 - 259, IFIP WG1.8 Workshop on Applying Concurrency Research in Industry [*DOI :* 10.1016/J.JLAP.2008.11.005], http://www.sciencedirect.com/science/article/pii/S1567832608000957

[4] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC, J.-P. TALPIN. *Polychronous Design of Embedded Real-Time Applications*, in "ACM Transactions on Software Engineering and Methodology (TOSEM)", April 2007, vol. 16, n^{o} 2, http://doi.acm.org/10.1145/1217295.1217298

[5] A. GAMATIÉ, T. GAUTIER. *The Signal Synchronous Multiclock Approach to the Design of Distributed Embedded Systems*, in "IEEE Transactions on Parallel and Distributed Systems", 2010, vol. 21, n^{o} 5, pp. 641-657 [*DOI :* 10.1109/TPDS.2009.125], http://hal.inria.fr/inria-00522794

[6] A. GAMATIÉ, T. GAUTIER, P. LE GUERNIC. *Synchronous design of avionic applications based on model refinements*, in "Journal of Embedded Computing (IOS Press)", 2006, vol. 2, n^{o} 3-4, pp. 273-289, http://hal.archives-ouvertes.fr/hal-00541523

[7] P. LE GUERNIC, J.-P. TALPIN, J.-C. LE LANN. *Polychrony for system design*, in "Journal of Circuits, Systems and Computers, Special Issue on Application Specific Hardware Design", June 2003, vol. 12, n^{o} 03, http://hal.inria.fr/docs/00/07/18/71/PDF/RR-4715.pdf

[8] D. POTOP-BUTUCARU, Y. SOREL, R. DE SIMONE, J.-P. TALPIN. *From Concurrent Multi-clock Programs to Deterministic Asynchronous Implementations*, in "Fundamenta Informaticae", January 2011, vol. 108, n^{o} 1-2, pp. 91–118, http://dl.acm.org/citation.cfm?id=2362088.2362094

[9] J.-P. TALPIN, J. OUY, T. GAUTIER, L. BESNARD, P. LE GUERNIC. *Compositional design of isochronous systems*, in "Science of Computer Programming", February 2012, vol. 77, n^{o} 2, pp. 113-128 [*DOI :* 10.1016/J.SCICO.2010.06.006], http://hal.archives-ouvertes.fr/hal-00768341

[10] H. YU, J. PRASHI, J.-P. TALPIN, S. K. SHUKLA, S. SHIRAISHI. *Model-Based Integration for Automotive Control Software*, in "Digital Automation Conference", San Francisco, United States, ACM, June 2015, https://hal.inria.fr/hal-01148905

Publications of the year

Articles in International Peer-Reviewed Journals

[11] T. GAUTIER, C. GUY, A. HONORAT, P. LE GUERNIC, J.-P. TALPIN, L. BESNARD. *Polychronous Automata and their Use for Formal Validation of AADL Models*, in "Frontiers of Computer Science", 2017, forthcoming [*DOI :* 10.1007/S11704-017-6134-5], https://hal.inria.fr/hal-01411257

[12] K. HU, T. ZHANG, L. SHANG, Z. YANG, J.-P. TALPIN. *Parallel Code Generation from Synchronous Specifications*, in "Journal of Software", November 2017, vol. 28, pp. 1-15 [*DOI :* 10.13328/J.CNKI.JOS.005056], https://hal.inria.fr/hal-01644290

Invited Conferences

[13] J.-P. TALPIN. *Compositional methods for CPS design (keynote abstract)*, in "3rd Symposium on Dependable Software Engineering: Theories, Tools and Applications", Changsha, China, Springer, October 2017, https://hal.inria.fr/hal-01615148

International Conferences with Proceedings

[14] A. HONORAT, H. N. TRAN, L. BESNARD, T. GAUTIER, J.-P. TALPIN, A. BOUAKAZ. *ADFG: a scheduling synthesis tool for dataflow graphs in real-time systems*, in "International Conference on Real-Time Networks and Systems", Grenoble, France, October 2017, pp. 1-10 [*DOI :* 10.1145/3139258.3139267], https://hal.inria.fr/hal-01615142

[15] S. LUNEL, B. BOYER, J.-P. TALPIN. *Compositional proofs in differential dynamic logic dL*, in "17th International Conference on Application of Concurrency to System Design", Zaragoza, Spain, June 2017, https://hal.inria.fr/hal-01615140

[16] O. SANKUR, J.-P. TALPIN. *An Abstraction Technique For Parameterized Model Checking of Leader Election Protocols: Application to FTSP*, in "23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)", Uppsala, Sweden, Lecture Notes in Computer Science, April 2017, vol. 10206, https://hal.archives-ouvertes.fr/hal-01431472

Scientific Books (or Scientific Book chapters)

[17] L. BESNARD, T. GAUTIER, P. LE GUERNIC, C. GUY, J.-P. TALPIN, B. LARSON, E. BORDE. *Formal Semantics of Behavior Specifications in the Architecture Analysis and Design Language Standard*, in "Cyber-Physical System Design from an Architecture Analysis Viewpoint", Cyber-Physical System Design from an Architecture Analysis Viewpoint, Springer, January 2017 [*DOI :* 10.1007/978-981-10-4436-6_3], https://hal.inria.fr/hal-01615143

[18] S. NAKAJIMA, J.-P. TALPIN, M. TOYOSHIMA, H. YU. *Cyber-Physical System Design from an Architecture Analysis Viewpoint*, Communications of NII Shonan Meetings, Springer, January 2017 [*DOI :* 10.1007/978-981-10-4436-6], https://hal.inria.fr/hal-01615144